SUBSTITUTE FORM PTO-1390 U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE	ATTORNEY'S DOCKET NUMBER 10559-305US1

TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371

U.S. APPLICATION NO. (If Known, see 37 CFR 1.5) **10/069229**

INTERNATIONAL APPLICATION NO. PCT/US00/23996	INTERNATIONAL FILING DATE 31 August 2000	PRIORITY DATE CLAIMED 1 September 1999

TITLE OF INVENTION
BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE

APPLICANT(S) FOR DO/EO/US
Gilbert Wolrich, Matthew J. Adiletta, William R. Wheeler, Debra Bernstein and Donald F. Hooper

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☒ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.

2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.

3. ☒ This is an express request to promptly begin national examination procedures (35 U.S.C. 371(f)).

4. ☒ The US has been elected by the expiration of 19 months from the priority date (PCT Article 31).

5. ☒ A copy of the International Application as filed (35 U.S.C. 371(c)(2))
    a. ☒ is attached hereto (required only if not communicated by the International Bureau).
    b. ☐ has been communicated by the International Bureau.
    c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US).

6. ☐ An English language translation of the International Application as filed (35 U.S.C. 371(c)(2)).

7. ☒ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371(c)(3))
    a. ☐ are attached hereto (required only if not communicated by the International Bureau).
    b. ☐ have been communicated by the International Bureau.
    c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
    d. ☒ have not been made and will not be made.

8. ☐ An English language translation of amendments to the claims under PCT Article 19 (35 U.S.C. 371(c)(3)).

9. ☐ An oath or declaration of the inventor(s) (35 U.S.C. 371(c)(4)).

10. ☐ An English language translation of the annexes to the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

1 to 16 below concern other documents or information included:

11. ☒ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.

12. ☐ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.

13. ☐ A FIRST preliminary amendment.
    ☐ A SECOND or SUBSEQUENT preliminary amendment.

14. ☐ A substitute specification.

15. ☐ A change of power of attorney and/or address letter.

16. ☐ Other items or information:

☐
☐
☐
☐
☐

CERTIFICATE OF MAILING BY EXPRESS MAIL   Express Mail Label No. EV044491983US

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, DC 20231.

2-20-02
Date of Deposit

Signature

Gabriel Lewis
Typed Name of Person Signing

U.S. APPLICATION NO. (IF KNOWN) 10/069229	INTERNATIONAL APPLICATION NO. PCT/US00/23996	ATTORNEY'S DOCKET NUMBER 10559-305US1

17. ☒ The following fees are submitted:

Basic National Fee (37 CFR 1.492(a)(1)- (5)):

	CALCULATIONS PTO USE ONLY

Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO............ **$1040**

International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO **$890**

International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO...................... **$740**

International preliminary examination fee paid to USPTO (37 CFR 1.482) but all claims did not satisfy provisions of PCT Article 33(1)-(4).................. **$710**

International preliminary examination fee paid to USPTO (37 CFR 1.482) and all claims satisfied provisions of PCT Article 33(1)-(4) **$100**

ENTER APPROPRIATE BASIC FEE AMOUNT =	$710.00	

Surcharge of **$130** for furnishing the oath or declaration later than ☐ 20 ☒ 30 months from the earliest claimed priority date (37 CFR 1.492(e)).

				$130.00	

Claims	Number Filed	Number Extra	Rate		
Total Claims	21 - 20 =	1	x **$18**	$18.00	
Independent Claims	4 - 3 =	1	x **$84**	$84.00	
MULTIPLE DEPENDENT CLAIMS(S) (if applicable)			+ **$280**	$0.00	
TOTAL OF ABOVE CALCULATIONS =				$942.00	

☐ Applicant claims small entity status. See 37 CFR 1.27. The fees indicated above are reduced by 1/2.

	$0.00	
SUBTOTAL =	$942.00	

Processing fee of **$130** for furnishing the English Translation later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(f))

	$0.00	
TOTAL NATIONAL FEE =	$942.00	

Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31).**$40.00** per property +

	$0.00	
TOTAL FEES ENCLOSED =	$942.00	
	Amount to be refunded:	$
	Charged:	$

a. ☒ A check in the amount of $942.00 to cover the above fees is enclosed.
b. ☐ Please charge my Deposit Account No. 06-1050 in the amount of $0.00 to cover the above fees. A duplicate copy of this sheet is enclosed.
c. ☒ The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. 06-1050. A duplicate copy of this sheet is enclosed.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137(a) or (b) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

Scott C. Harris
FISH & RICHARDSON P.C.
4350 La Jolla Village Drive, Suite 500
San Diego, California 92122
(858) 678-5070 *phone*
(858) 678-5099 *facsimile*

SIGNATURE :

NAME   Scott C. Harris

REGISTRATION NUMBER   32,030

10166142.doc

BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE

BACKGROUND

This invention relates to branch instructions.

5   Parallel processing is an efficient form of information processing of concurrent events in a computing process. Parallel processing demands concurrent execution of many programs in a computer. Sequential processing or serial processing has all tasks performed sequentially at a single station whereas, pipelined processing has tasks performed at specialized stations. Computer code whether executed in parallel

10   processing, pipelined or sequential processing machines involves branches in which an instruction stream may execute in a sequence and branch from the sequence to a different sequence of instructions.

BRIEF DESCRIPTION OF THE DRAWINGS

15   FIG. 1 is a block diagram of a communication system employing a processor.

FIG. 2 is a detailed block diagram of the processor.

FIG. 3 is a block diagram of a microengine used in the processor of FIGS. 1 and 2.

20   FIG. 4 is a diagram of a pipeline in the microengine.

FIG. 5 shows exemplary formats for branch instructions.

FIG. 6 is a block diagram of general purpose registers.

DESCRIPTION

25   Referring to FIG. 1, a communication system 10 includes a processor 12. In one embodiment, the processor is a hardware-based multithreaded processor 12. The processor 12 is coupled to a bus such as a PCI bus 14, a memory system 16 and a second bus 18. The system 10 is especially useful for tasks that can be broken into parallel sub-tasks or functions. Specifically hardware-based multithreaded processor 12 is useful for

30   tasks that are bandwidth oriented rather than latency oriented. The hardware-based multithreaded processor 12 has multiple microengines 22 each with multiple hardware controlled threads that can be simultaneously active and independently work on a task.

The hardware-based multithreaded processor 12 also includes a central controller 20 that assists in loading microcode control for other resources of the hardware-based multithreaded processor 12 and performs other general purpose computer type functions such as handling protocols, exceptions, extra support for packet processing

5   where the microengines pass the packets off for more detailed processing such as in boundary conditions. In one embodiment, the processor 20 is a Strong Arm® (Arm is a trademark of ARM Limited, United Kingdom) based architecture. The general purpose microprocessor 20 has an operating system. Through the operating system the processor 20 can call functions to operate on microengines 22a-22f. The processor 20 can use any

10   supported operating system preferably a real time operating system. For the core processor implemented as a Strong Arm architecture, operating systems such as, MicrosoftNT® real-time, VXWorks and □CUS, a freeware operating system available over the Internet, can be used.

The hardware-based multithreaded processor 12 also includes a plurality

15   of function microengines 22a-22f. Functional microengines (microengines) 22a-22f each maintain a plurality of program counters in hardware and states associated with the program counters. Effectively, a corresponding plurality of sets of threads can be simultaneously active on each of the microengines 22a-22f while only one is actually operating at any one time.

20   Microengines 22a-22f each have capabilities for processing four hardware threads. The microengines 22a-22f operate with shared resources including memory system 16 and bus interfaces 24 and 28. The memory system 16 includes a Synchronous Dynamic Random Access Memory (SDRAM) controller 26a and a Static Random Access Memory (SRAM) controller 26b. SDRAM memory 16a and SDRAM controller 26a are

25   typically used for processing large volumes of data, e.g., processing of network payloads from network packets. The SRAM controller 26b and SRAM memory 16b are used in, e.g., networking packet processing, postscript processor, or as a processor for a storage subsystem, i.e., RAID disk storage, or for low latency, fast access tasks, e.g., accessing look-up tables, memory for the core processor 20, and so forth.

30   The processor 12 includes a bus interface 28 that couples the processor to the second bus 18. Bus interface 28 in one embodiment couples the processor 12 to the so-called FBUS 18 (FIFO bus). The processor 12 includes a second interface e.g., a PCI bus interface 24 that couples other system components that reside on the PCI 14 bus to

the processor 12. The PCI bus interface 24, provides a high speed data path 24a to the SDRAM memory 16a. Through that path data can be moved quickly from the SDRAM 16a through the PCI bus 14, via direct memory access (DMA) transfers.

Each of the functional units are coupled to one or more internal buses.
5   The internal buses are dual, 32 bit buses (i.e., one bus for read and one for write). The hardware-based multithreaded processor 12 also is constructed such that the sum of the bandwidths of the internal buses in the processor 12 exceed the bandwidth of external buses coupled to the processor 12. The processor 12 includes an internal core processor bus 32, e.g., an ASB bus (Advanced System Bus) that couples the processor core 20 to
10   the memory controller 26a, 26c and to an ASB translator 30 described below. The ASB bus is a subset of the so called AMBA bus that is used with the Strong Arm processor core. The processor 12 also includes a private bus 34 that couples the microengine units to SRAM controller 26b, ASB translator 30 and FBUS interface 28. A memory bus 38 couples the memory controller 26a, 26b to the bus interfaces 24 and 28 and memory
15   system 16 including flashrom 16c used for boot operations and so forth.

Referring to FIG. 2, each of the microengines 22a-22f includes an arbiter that examines flags to determine the available threads to be operated upon. Any thread from any of the microengines 22a-22f can access the SDRAM controller 26a, SDRAM controller 26b or FBUS interface 28. The memory controllers 26a and 26b each include a
20   plurality of queues to store outstanding memory reference requests. The FBUS interface 28 supports Transmit and Receive flags for each port that a MAC device supports, along with an Interrupt flag indicating when service is warranted. The FBUS interface 28 also includes a controller 28a that performs header processing of incoming packets from the FBUS 18. The controller 28a extracts the packet headers and performs a
25   microprogrammable source/destination/protocol hashed lookup (used for address smoothing) in SRAM.

The core processor 20 accesses the shared resources. The core processor 20 has a direct communication to the SDRAM controller 26a to the bus interface 24 and to SRAM controller 26b via bus 32. However, to access the microengines 22a-22f and
30   transfer registers located at any of the microengines 22a-22f, the core processor 20 access the microengines 22a-22f via the ASB Translator 30 over bus 34. The ASB translator 30 can physically reside in the FBUS interface 28, but logically is distinct. The ASB Translator 30 performs an address translation between FBUS microengine transfer

register locations and core processor addresses (i.e., ASB bus) so that the core processor 20 can access registers belonging to the microengines 22a-22c.

Although microengines 22 can use the register set to exchange data as described below, a scratchpad memory 27 is also provided to permit microengines to

5   write data out to the memory for other microengines to read. The scratchpad 27 is coupled to bus 34.

The processor core 20 includes a RISC core 50 implemented in a five stage pipeline performing a single cycle shift of one operand or two operands in a single cycle, provides multiplication support and 32 bit barrel shift support. This RISC core 50

10   is a standard Strong Arm® architecture but it is implemented with a five stage pipeline for performance reasons. The processor core 20 also includes a 16 kilobyte instruction cache 52, an 8 kilobyte data cache 54 and a prefetch stream buffer 56. The core processor 20 performs arithmetic operations in parallel with memory writes and instruction fetches. The core processor 20 interfaces with other functional units via the ARM defined ASB

15   bus. The ASB bus is a 32-bit bi-directional bus 32.

Referring to FIG. 3, an exemplary microengine 22f includes a control store 70 that includes a RAM which stores a microprogram. The microprogram is loadable by the core processor 20. The microengine 22f also includes controller logic 72. The controller logic includes an instruction decoder 73 and program counter (PC) units 72a-

20   72d. The four micro program counters 72a-72d are maintained in hardware. The microengine 22f also includes context event switching logic 74. Context event logic 74 receives messages (e.g., SEQ_#_EVENT_RESPONSE; FBI_EVENT_RESPONSE; SRAM _EVENT_RESPONSE; SDRAM _EVENT_RESPONSE; and ASB _EVENT_RESPONSE) from each one of the shared resources, e.g., SRAM 26a, SDRAM

25   26b, or processor core 20, control and status registers, and so forth. These messages provide information on whether a requested function has completed. Based on whether or not a function requested by a thread has completed and signaled completion, the thread needs to wait for that completion signal, and if the thread is enabled to operate, then the thread is placed on an available thread list (not shown). The microengine 22f can have a

30   maximum of e.g., 4 threads available.

In addition to event signals that are local to an executing thread, the microengines 22 employ signaling states that are global. With signaling states, an executing thread can broadcast a signal state to all microengines 22. Receive Request or

4

Available signal, any and all threads in the microengines can branch on these signaling states. These signaling states can be used to determine availability of a resource or whether a resource is due for servicing.

The context event logic 74 has arbitration for the four (4) threads. In one

5   embodiment, the arbitration is a round robin mechanism. Other techniques could be used including priority queuing or weighted fair queuing. The microengine 22f also includes an execution box (EBOX) data path 76 that includes an arithmetic logic unit 76a and general purpose register set 76b. The arithmetic logic unit 76a performs arithmetic and logical functions as well as shift functions. The arithmetic logic unit includes condition

10   code bits that are used by instructions described below. The registers set 76b has a relatively large number of general purpose registers that are windowed as will be described so that they are relatively and absolutely addressable.   The microengine 22f also includes a write transfer register stack 78 and a read transfer stack 80. These registers are also windowed so that they are relatively and absolutely addressable. Write transfer

15   register stack 78 is where write data to a resource is located. Similarly, read register stack 80 is for return data from a shared resource. Subsequent to or concurrent with data arrival, an event signal from the respective shared resource e.g., the SRAM controller 26a, SDRAM controller 26b or core processor 20 will be provided to context event arbiter 74 which will then alert the thread that the data is available or has been sent. Both

20   transfer register banks 78 and 80 are connected to the execution box (EBOX) 76 through a data path.

Referring to FIG. 4, the microengine datapath maintains a 5-stage micro-pipeline 82. This pipeline includes lookup of microinstruction words 82a, formation of the register file addresses 82b, read of operands from register file 82c, ALU, shift or

25   compare operations 82d, and write-back of results to registers 82e. By providing a write-back data bypass into the ALU/shifter units, and by assuming the registers are implemented as a register file (rather than a RAM), the microengine can perform a simultaneous register file read and write, which completely hides the write operation.

The instruction set supported in the microengines 22a-22f support

30   conditional branches. The worst case conditional branch latency (not including jumps) occurs when the branch decision is a result of condition codes being set by the previous microcontrol instruction. The latency is shown below in Table 1:

TABLE 1

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-----------------+----+----+----+----+----+----+----+----+
microstore lookup | n1 | cb | n2 | XX | b1 | b2 | b3 | b4 |
reg addr gen      |    | n1 | cb | XX | XX | b1 | b2 | b3 |
reg file lookup   |    |    | n1 | cb | XX | XX | b1 | b2 |
ALU/shifter/cc    |    |    |    | n1 | cb | XX | XX | b1 |
write back        |    |    | m2|    | n1 | cb | XX | XX |
```

where nx is pre-branch microword (n1 sets cc's), cb is conditional branch, bx is post-branch microword and XX is an aborted microword

As shown in Table 1, it is not until cycle 4 that the condition codes of n1 are set, and the branch decision can be made (which in this case causes the branch path to be looked up in cycle 5). The microengine 22f incurs a 2-cycle branch latency penalty because it must abort operations n2 and n3 (the 2 microwords directly after the branch) in the pipe, before the branch path begins to fill the pipe with operation b1. If the branch is not taken, no microwords are aborted and execution continues normally. The microengines have several mechanisms to reduce or eliminate the effective branch latency.

The microengines support selectable deferred branches. Selectable deferring branches are when a microengine allows 1 or 2 micro instructions after the branch to execute before the branch takes effect (i.e. the effect of the branch is "deferred" in time). Thus, if useful work can be found to fill the wasted cycles after the branch microword, then the branch latency can be hidden. A 1-cycle deferred branch is shown below in Table 2 where n2 is allowed to execute after cb, but before b1:

TABLE 2

```
                 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
------------------+----+----+----+----+----+----+----+----+
microstore lookup | n1 | cb | n2 | XX | b1 | b2 | b3 | b4 |
reg addr gen      |    | n1 | cb | n2 | XX | b1 | b2 | b3 |
reg file lookup   |    |    | n1 | cb | n2 | XX | b1 | b2 |
ALU/shifter/cc    |    |    |    | n1 | cb | n2 | XX | b1 |
write back        |    |    |    |    | n1 | cb | n2 | XX |
```

A 2-cycle deferred branch is shown in TABLE 3 where n2 and n3 are both allowed to complete before the branch to b1 occurs. Note that a 2-cycle branch deferment is only allowed when the condition codes are set on the microword preceding the branch.

TABLE 3

```
                 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
------------------+----+----+----+----+----+----+----+----+----+
microstore lookup | n1 | cb | n2 | n3 | b1 | b2 | b3 | b4 | b5 |
reg addr gen      |    | n1 | cb | n2 | n3 | b1 | b2 | b3 | b4 |
reg file lkup     |    |    | n1 | cb | n2 | n3 | b1 | b2 | b3 |
ALU/shftr/cc      |    |    |    | n1 | cb | n2 | n3 | b1 | b2 |
write back        |    |    |    |    | n1 | cb | n2 | n3 | b1 |
```

The microengines also support condition code evaluation. If the condition codes upon which a branch decision are made are set 2 or more microwords before the branch, then 1 cycle of branch latency can be eliminated because the branch decision can be made 1 cycle earlier as in Table 4.

TABLE 4

```
                     | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
----------------------+----+----+----+----+----+----+----+----+
microstore lookup | n1 | n2 | cb | XX | b1 | b2 | b3 | b4 |
reg addr gen      |    | n1 | n2 | cb | XX | b1 | b2 | b3 |
reg file lookup   |    |    | n1 | n2 | cb | XX | b1 | b2 |
ALU/shifter/cc    |    |    |    | n1 | n2 | cb | XX | b1 |
write back        |    |    |    |    | n1 | n2 | cb | XX |
```

In this example, n1 sets the condition codes and n2 does not set the conditions codes. Therefore, the branch decision can be made at cycle 4 (rather than 5), to eliminate 1 cycle of branch latency. In the example in Table 5 the 1-cycle branch deferment and early setting of condition codes are combined to completely hide the branch latency. That is, the condition codes (cc's) are set 2 cycles before a 1-cycle deferred branch.

TABLE 5

```
                     | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
----------------------+----+----+----+----+----+----+----+----+
microstore lookup | n1 | n2 | cb | n3 | b1 | b2 | b3 | b4 |
reg addr gen      |    | n1 | n2 | cb | n3 | b1 | b2 | b3 |
reg file lookup   |    |    | n1 | n2 | cb | n3 | b1 | b2 |
ALU/shifter/cc    |    |    |    | n1 | n2 | cb | n3 | b1 |
write back        |    |    |    |    | n1 | n2 | cb | n3 |
```

In the case where the condition codes cannot be set early (i.e. they are set in the microword preceding the branch), the microengine supports branch guessing which attempts to reduce the 1 cycle of exposed branch latency that remains. By "guessing" the branch path or the sequential path, the microsequencer pre-fetches the guessed path 1 cycle before it definitely knows what path to execute. If it guessed correctly, 1 cycle of branch latency is eliminated as shown in Table 6.

TABLE 6

guess branch taken /branch is taken

```
                     | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
--------------------+----+----+----+----+----+----+----+----+
microstore lookup   | n1 | cb | n1 | b1 | b2 | b3 | b4 | b5 |
reg addr gen        |    | n1 | cb | XX | b1 | b2 | b3 | b4 |
reg file lookup     |    |    | n1 | cb | XX | b1 | b2 | b3 |
ALU/shifter/cc      |    |    |    | n1 | cb | XX | b1 | b2 |
write back          |    |    |    |    | n1 | cb | XX | b1 |
```

If the microcode guessed a branch taken incorrectly, the microengine still only wastes 1 cycle as in TABLE 7

TABLE 7

guess branch taken /branch is NOT taken

```
                     | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
--------------------+----+----+----+----+----+----+----+----+
microstore lookup   | n1 | cb | n1 | XX | n2 | n3 | n4 | n5 |
reg addr gen        |    | n1 | cb | n1 | XX | n2 | n3 | n4 |
reg file lookup     |    |    | n1 | cb | n1 | XX | n2 | n3 |
ALU/shifter/cc      |    |    |    | n1 | cb | n1 | XX | n2 |
write back          |    |    |    |    | n1 | cb | n1 | XX |
```

However, the latency penalty is distributed differently when microcode guesses a branch is not taken. For guess branch NOT taken / branch is NOT taken there are no wasted cycles as in Table 8.

Table 8

```
                    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
--------------------+----+----+----+----+----+----+----+----+
microstore lookup  | n1 | cb | n1 | n2 | n3 | n4 | n5 | n6 |
reg addr gen       |    | n1 | cb | n1 | n2 | n3 | n4 | n5 |
reg file lookup    |    |    | n1 | cb | n1 | n2 | n1 | b4 |
ALU/shifter/cc     |    |    |    | n1 | cb | n1 | n2 | n3 |
write back         |    |    |    |    | n1 | cb | n1 | n2 |
```

However for guess branch NOT taken /branch is taken there are 2 wasted cycles as in Table 9.

Table 9

```
                    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
--------------------+----+----+----+----+----+----+----+----+
microstore lookup  | n1 | cb | n1 | XX | b1 | b2 | b3 | b4 |
reg addr gen       |    | n1 | cb | XX | XX | b1 | b2 | b3 |
reg file lookup    |    |    | n1 | cb | XX | XX | b1 | b2 |
ALU/shifter/cc     |    |    |    | n1 | cb | XX | XX | b1 |
write back         |    |    |    |    | n1 | cb | XX | XX |
```

The microengine can combine branch guessing with 1-cycle branch deferment to improve the result further. For guess branch taken with 1-cycle deferred branch/branch is taken is in Table 10.

Table 10

```
                   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
-------------------+----+----+----+----+----+----+----+----+
microstore lookup  | n1 | cb | n2 | b1 | b2 | b3 | b4 | b5 |
reg addr gen       |    | n1 | cb | n2 | b1 | b2 | b3 | b4 |
reg file lookup    |    |    | n1 | cb | n2 | b1 | b2 | b3 |
ALU/shifter/cc     |    |    |    | n1 | cb | n2 | b1 | b2 |
write back         |    |    |    |    | n1 | cb | n2 | b1 |
```

In the case above, the 2 cycles of branch latency are hidden by the execution of n2, and by correctly guessing the branch direction.

If microcode guesses incorrectly, 1 cycle of branch latency remains exposed as in Table 11 (guess branch taken with 1-cycle deferred branch/branch is NOT taken).

Table 11

```
                   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
-------------------+----+----+----+----+----+----+----+----+----+
microstore lookup  | n1 | cb | n2 | XX | n3 | n4 | n5 | n6 | n7 |
reg addr gen       |    | n1 | cb | n2 | XX | n3 | n4 | n5 | n6 |
reg file lkup      |    |    | n1 | cb | n2 | XX | n3 | n4 | n5 |
ALU/shftr/cc       |    |    |    | n1 | cb | n2 | XX | n3 | n4 |
write back         |    |    |    |    | n1 | cb | n2 | XX | n3 |
```

If microcode correctly guesses a branch NOT taken, then the pipeline flows sequentially in the normal unperturbed case. If microcode incorrectly guesses branch NOT taken, the microengine again exposes 1 cycle of unproductive execution as shown in Table 12.

Table 12

guess branch NOT taken/branch is taken

	1	2	3	4	5	6	7	8	9
microstore lookup	n1	cb	n2	XX	b1	b2	b3	b4	b5
reg addr gen		n1	cb	n2	XX	b1	b2	b3	b4
reg file lkup			n1	cb	n2	XX	b1	b2	b3
ALU/shftr/cc				n1	cb	n2	XX	b1	b2
write back					n1	cb	n2	XX	b1

where nx is pre-branch microword (n1 sets cc's)

cb is conditional branch

bx is post-branch microword

XX is aborted microword

In the case of a jump instruction, 3 extra cycles of latency are incurred because the branch address is not known until the end of the cycle in which the jump is in the ALU stage (Table 13).

Table 13

	1	2	3	4	5	6	7	8	9
microstore lookup	n1	jp	XX	XX	XX	j1	j2	j3	j4
reg addr gen		n1	jp	XX	XX	XX	j1	j2	j3
reg file lkup			n1	jp	XX	XX	XX	j1	j2
ALU/shftr/cc				n1	jp	XX	XX	XX	j1
write back					n1	jp	XX	XX	XX

Referring to FIG. 5, the microengines support various branch instructions such as those that branch on condition codes. In addition, the microengines also support a branch instruction that branches on a byte being equal or not equal to a specified byte. The branch on byte instruction "BR=BYTE" includes a byte_spec field.

5　　BR=BYTE, BR!=BYTE

This branch instruction branches to an instruction at the specified label if a specified byte in a longword matches or mismatches the byte_compare_value. The br=byte instruction prefetches the instruction for the "branch taken" condition rather than the next sequential instruction. The br!=byte instruction prefetches the next sequential

10　　instruction. These instructions set the condition codes in the microengine.

Format: br=byte[reg, byte_spec, byte_compare_value, label#], optional_token

br!=byte[reg, byte_spec, byte_compare_value, label#], optional_token

15　　　　　　　　Reg A is a context-relative transfer register or general-purpose register that holds the operand. Byte_spec Number specifies a byte in register to be compared with byte_compare_value. Valid byte_spec values are 0 through 3. A value of 0 refers to the rightmost byte.

A byte_compare_value is a value used for comparison. Valid

20　　byte_compare_values are 0 to 255. The label# is a symbolic label corresponding to the address of an instruction. The instruction also can include an optional_token. In this example the optional token can be a defer one value which will execute one instruction following this branch instruction before performing the branch operation. The optional token can alternatively be a defer two instructions which is allowed with the br!=byte

25　　instruction and executes the two instructions following this branch instruction before performing the branch operation. The optional token can alternatively be a defer three instructions which is allowed with the br!=byte instruction and which causes the processor to execute the three instructions following this branch instruction before performing the branch operation.

30　　　　　　　　Example: br!=byte[reg, byte_spec, byte_compare_value, label#], defer[3]

This instruction represents an instruction that compares an aligned byte of a register operand to an immediate specified byte value. The byte_spec parameter

represents the aligned byte to compare (0 is right-most byte, 3 is leftmost byte). The ALU condition codes are set by subtracting the specified byte value from the specified register byte. If the values match, the specified branch is taken. There is a 3 cycle branch latency associated with this microword, since condition codes are evaluated and therefore, the

5   branch deferments of 0, 1, 2 or 3 are allowed in order to fill the latency with useful work. The register can be an A or B bank register.

Referring to FIG. 6, the two register address spaces that exist are Locally accessibly registers, and Globally accessible registers accessible by all microengines. The General Purpose Registers (GPRs) are implemented as two separate banks (A bank

10   and B bank) whose addresses are interleaved on a word-by-word basis such that A bank registers have lsb=0, and B bank registers have lsb=1. Each bank is capable of performing a simultaneous read and write to two different words within its bank.

Across banks A and B, the register set 76b is also organized into four windows $76b_0$-$76b_3$ of 32 registers that are relatively addressable per thread. Thus,

15   thread_0 will find its register 0 at 77a (register 0), the thread_1 will find its register_0 at 77b (register 32), thread_2 will find its register_0 at 77c (register 64), and thread_3 at 77d (register 96). Relative addressing is supported so that multiple threads can use the exact same control store and locations but access different windows of register and perform different functions. The uses of register window addressing and bank addressing provide

20   the requisite read bandwidth using only dual ported RAMS in the microengine 22f.

These windowed registers do not have to save data from context switch to context switch so that the normal push and pop of a context swap file or stack is eliminated. Context switching here has a 0 cycle overhead for changing from one context to another. Relative register addressing divides the register banks into windows across

25   the address width of the general purpose register set. Relative addressing allows access any of the windows relative to the starting point of the window. Absolute addressing is also supported in this architecture where any one of the absolute registers may be accessed by any of the threads by providing the exact address of the register.

Addressing of general purpose registers 78 can occur in 2 modes

30   depending on the microword format. The two modes are absolute and relative. In absolute mode, addressing of a register address is directly specified in 7-bit source field (a6-a0 or b6-b0):

14

```
                7 6 5 4 3 2 1 0
                +---+---+---+---+---+---+---+---+
      A GPR:   | a6| 0 | a5| a4| a3| a2| a1| a0|   a6=0
      B GPR:   | b6| 1 | b5| b4| b3| b2| b1| b0|   b6=0
      SRAM/ASB:| a6| a5| a4| 0 | a3| a2| a1| a0|   a6=1, a5=0, a4=0    SDRAM:
               | a6| a5| a4| 0 | a3| a2| a1| a0|   a6=1, a5=0, a4=1
```

register address directly specified in 8-bit dest field (d7-d0):

```
                7 6 5 4 3 2 1 0
                +---+---+---+---+---+---+---+---+
      A GPR:    | d7| d6| d5| d4| d3| d2| d1| d0|  d7=0, d6=0
      B GPR:    | d7| d6| d5| d4| d3| d2| d1| d0|  d7=0, d6=1
      SRAM/ASB:| d7| d6| d5| d4| d3| d2| d1| d0|  d7=1, d6=0, d5=0
      SDRAM:    | d7| d6| d5| d4| d3| d2| d1| d0|  d7=1, d6=0, d5=1
```

If $<a6:a5>=1,1$, $<b6:b5>=1,1$, or $<d7:d6>=1,1$ then the lower bits are interpreted as a context-relative address field (described below). When a non-relative A or B source address is specified in the A, B absolute field, only the lower half of the SRAM/ASB and SDRAM address spaces can be addressed. Effectively, reading absolute SRAM/SDRAM devices has the effective address space; however, since this restriction does not apply to the dest field, writing the SRAM/SDRAM still uses the full address space.

In relative mode, addresses a specified address is offset within context space as defined by a 5-bit source field   (a4-a0 or b4-b0):

```
                  7  6  5  4  3  2  1  0
                +---+---+---+---+---+---+---+---+
    A GPR:    | a4| 0 |context| a3| a2| a1| a0| a4=0
    B GPR:    | b4| 1 |context| b3| b2| b1| b0| b4=0
    SRAM/ASB:|ab4| 0 |ab3|context| b2| b1|ab0| ab4=1, ab3=0
    SDRAM:   |ab4| 0 |ab3|context| b2| b1|ab0| ab4=1, ab3=1
```

or as defined by the 6-bit dest field (d5-d0):

```
                  7  6  5  4  3  2  1  0
                +---+---+---+---+---+---+---+---+
    A GPR:    | d5| d4|context| d3| d2| d1| d0|   d5=0, d4=0
    B GPR:    | d5| d4|context| d3| d2| d1| d0|   d5=0, d4=1
    SRAM/ASB:| d5| d4| d3|context| d2| d1| d0|   d5=1, d4=0, d3=0
    SDRAM:   | d5| d4| d3|context| d2| d1| d0|   d5=1, d4=0, d3=1
```

If $<d5:d4>=1,1$, then the destination address does not address a valid register, thus, no dest operand is written back.

Other embodiments are within the scope of the appended claims.

What is claimed is:

1.   A computer instruction, comprises:

        a branch instruction that causes a processor to branch from executing a first sequential series of instructions to a different sequential series of instructions based on a byte in a register being equal or not equal to a specifed byte value, if the specified

5   byte matches or mismatches the byte value.

2.   The instruction of claim 1 wherein the branch is to an instruction at a specified label.

10   3.   The instruction of claim 1 wherein the branch instruction comprises:

        a bit_postion field that specifies the byte in a longword contained in the register.

4.   The instruction of claim 1 wherein the branch instruction comprises:

15   an optional token that is set by a programmer and specifies a number i of instructions to execute following the branch instruction before performing the branch operation.

5.   The instruction of claim 1 wherein the branch instruction comprises:

20   an optional token that is set by a programmer and specifies a number i of instructions to execute following the branch instruction before performing the branch operation where the number of instructions can be specified as one, two or three.

6.   The instruction of claim 1 wherein the register is a context-relative transfer

25   register or a general-purpose register that holds the operand.

7.   The instruction of claim 1 wherein the branch instruction comprises:

        an optional token that is set by a programmer and which specifies a guess_branch prefetch for the instruction for the "branch taken" condition rather than the

30   next sequential instruction.

8.   The instruction of claim 1 wherein the branch instruction comprises:

an optional token that is set by a programmer and specifies a number i of instructions to execute following the branch instruction before performing the branch operation; and

a second optional token that is set by a programmer and which specifies a guess_branch prefetch for the instruction for the "branch taken" condition rather than the next sequential instruction.

9.   The instruction of claim 1 wherein the branch instruction allows a programmer to which bit of the register to use to determine the branch operation.

10.   The instruction of claim 1 wherein the branch instructions allows branches to occur based on evaluation of a byte that is in a data path of a processor.

11.   The instruction of claim 1 wherein the branch instruction branches on a byte matching the byte value and wherein the instruction prefetches the instruction for the "branch taken" condition.

12.   The instruction of claim 1 wherein the branch instruction branches on a byte not matching the byte value and wherein the instruction prefetches the next sequential instruction.

13.   The instruction of claim 1 wherein the branch instruction includes a Byte_spec Number that specifies the byte in the register to be compared with byte_compare_value.

14.   A computer program product residing on a computer readable medium for causing a processor that executes multiple contexts to perform a function comprises instructions causing the processor to:

            fetch a byte stored in a register;

5   determine whether the byte in the register is equal or not equal to a specifed byte value contained in the instruction; and

            perform a branching operation specified by the branch instruction based on the specified byte being equal or not equal to the byte in the register

10   15.   The product of claim 14 wherein the branch is to an instruction at a specified label.

16.   The product of claim 14 wherein the program includes a branch instruction that comprises:

15   a bit_postion field that specifies the byte in a longword contained in the register.

17.   A processor comprises:
            a register stack;
20   an arithmetic logic unit coupled to the register stack and a program control store that stores a branch instruction that causes the processor to:
            fetch a byte stored in a register;
            determine whether the byte in the register is equal or not equal to a specifed byte value contained in the instruction; and
25   perform a branching operation specified by the branch instruction based on the specified byte being equal or not equal to the byte in the register.
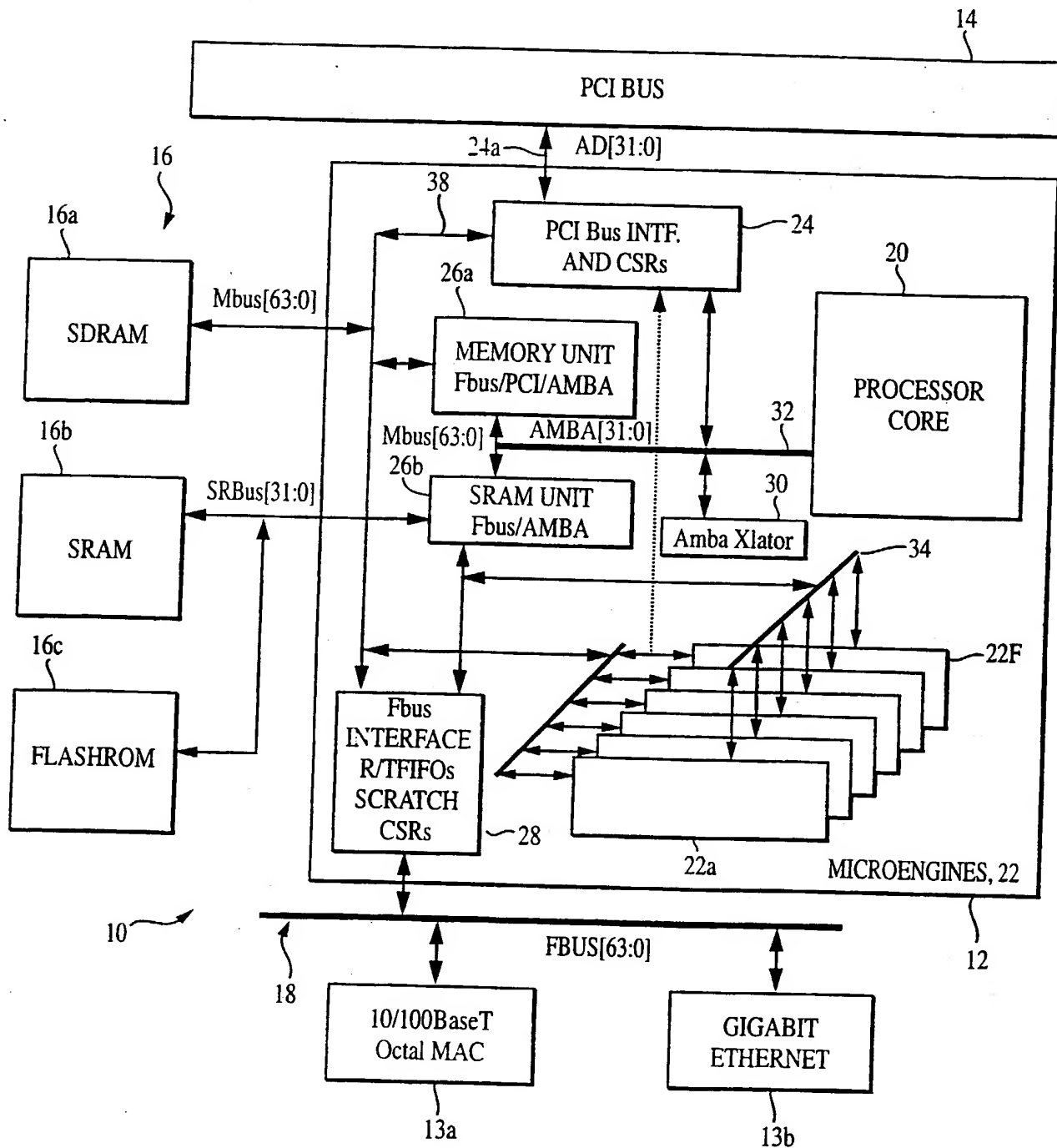
18.   The processor of claim 17 wherein instructions to perform the branch branch to an instruction at a specified label.

30

19.   The processor of claim 1 wherein a bit_postion field in an instruction specifies the byte in a longword contained in the register.

20.   A method of operating a processor comprises:

      fetching a byte stored in a register;

      determining whether the byte in the register is equal or not equal to a specifed byte value contained in the instruction; and

5   performing a branching operation specified by the branch instruction based on the specified byte being equal or not equal to the byte in the register.

21.   The method of claim 20 wherein performing the branch branches to an instruction at a specified label.

ABSTRACT

A processor such as a parallel hardware-based multithreaded processor (12) is described. The processor (12) can execute a computer instruction that is a branch instruction that causes the instruction sequence in the processor (12) to determine whether a byte in a register (80,78,76b) is equal or not equal to a specified byte value contained in the instruction and performing the branching operation specified by the branch instruction based on the specified byte being equal to the byte in the register (80,78,76b).

10166135.doc

1/7

FIG. 1

Matter No.: 10559-305US1
Page 2 of 7

WO 01/16716
BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE

10/069229

PCT/US00/23996

2/7

FIG. 2-1

FIG. 2-2

FIG. 2

FIG. 2-1

Matter No.: 10559-305US1   Page 3 of 7
Applicant(s): Gilbert Wolrich et al.
BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE

WO 01/16716   PCT/US00/23996

3/7

FIG. 2-2

Matter No.: 10559-305US1
Applicant(s): Gilbert Wolrich et al.
BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE

Page 4 of 7

100692291 0/069229

4/7

FIG. 3

Matter No.: 10559-305US1
Page 5 of 7
10/069229
Applicant(s): Gilbert Wolrich et al.
BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE
WO 01/16716
PCT/US00/23996

5/7

```
┌─────────────────────────┐
│   82a, LOOKUP           │
│   MICROINSTRUCTIONS     │
└─────────────────────────┘
            │
            ▼
┌─────────────────────────┐
│   82b, FORM REGISTER    │
│   FILE ADDRESSES        │
└─────────────────────────┘
            │
            ▼
┌─────────────────────────┐
│   82c, FETCH OPERANDS   │
└─────────────────────────┘
            │
            ▼
┌─────────────────────────┐
│   82d, ALU OPERATION    │
└─────────────────────────┘
            │
            ▼
┌─────────────────────────┐
│   82d, WRITE-BACK       │
│   OF RESULTS            │
└─────────────────────────┘
```

FIG. 4

10/069229

6/7

```
      31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
      +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
BR_BYTE | 0| 1| 1| 1|BShft|A/B Rel Src(aluB)|Hi-Im Data|       Branch Address       |defbr|Eq|Low-Im data|B-A
(set cc) +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

      +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
BR_REG | 0| 1| 1| 0|A/B Abs Src  (for ALUB)| xxxxxxxxx|       Immediate Data       |defbr|xxxxxxxxxxxxxxx|add
      +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Branch Descriptions:

Defbr => gives number of defered instructions (must be less/equal max_allowed (or BR_ev field))

Defered instructions can NOT be branch instructions

Br_Byte => (EQ assumes GT, NEQ assumes NT)

Can have defer = 3 on conditional branchs following br-bit or br-byte

FIG. 5

10/069229

7/7
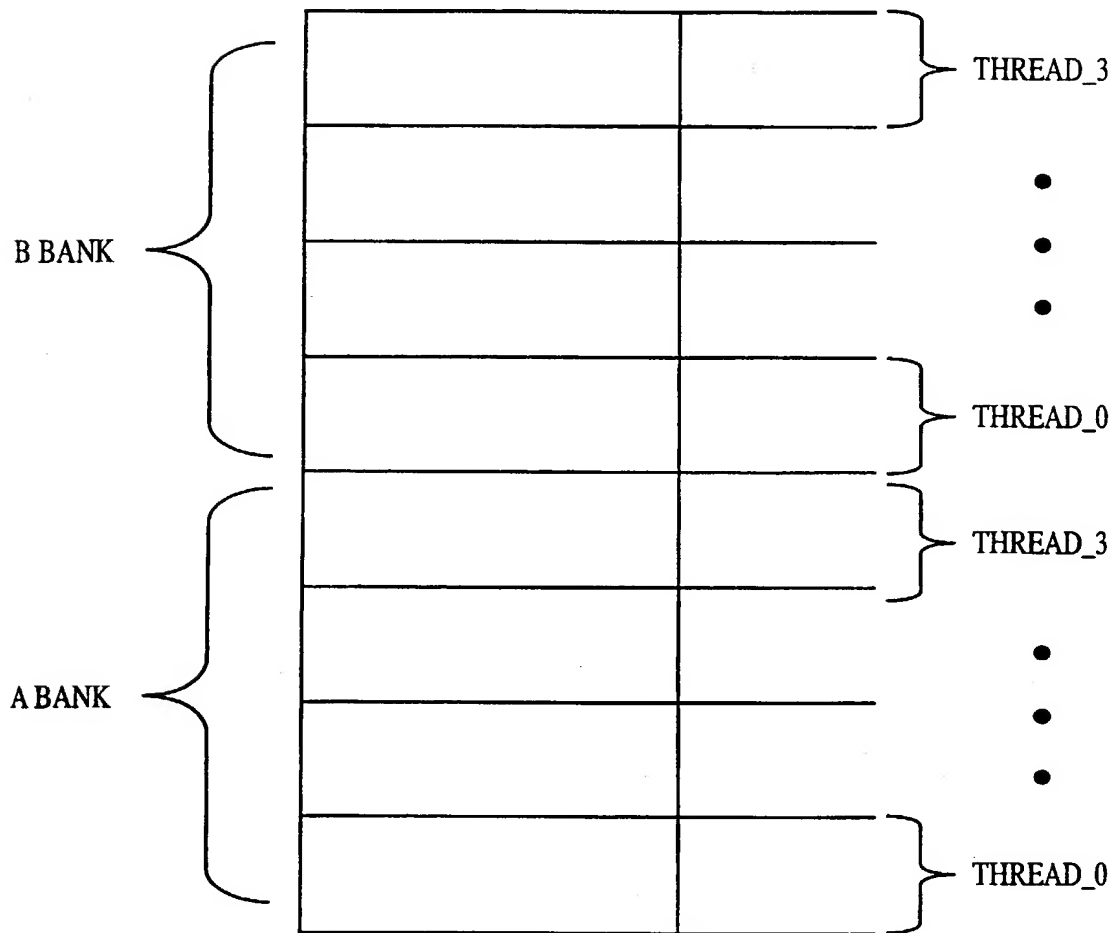
FIG. 6

COMBINED DECLARATION AND POWER OF ATTORNEY

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled <u>BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE</u>, the specification of which:

[]   is attached hereto.
[X]   was filed on <u>February 22, 2002</u> as Application Serial No. <u>10/069,229</u> and was amended on

_____.

[]   was described and claimed in PCT International Application No. _____ filed on
       _____ and as amended under PCT Article 19 on _____.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose all information I know to be material to patentability in accordance with Title 37, Code of Federal Regulations, §1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate or of any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed:

Country	Application No.	Filing Date	Priority Claimed
WIPO	PCT/US00/23996	08/31/2000	[X] Yes [] No
			[] Yes [] No

I hereby appoint the following attorneys and/or agents to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith:

Scott C. Harris, Reg. No. 32,030   Denis G. Maloney, Reg. No. 29,670
David L. Feigenbaum, Reg. No. 30,378   Kenneth F. Kozik, Reg. No. 36,572
Christina Sperry, Reg. No. 47,106   Paul A. Pysher, Reg. No. 40,780
Cathy L. Peterson, Reg. No. 41,249

Address all telephone calls to DENIS G. MALONEY at telephone number (617) 542-5070.

Address all correspondence to DENIS G. MALONEY at:

FISH & RICHARDSON P.C.
225 Franklin Street
Boston, Massachusetts 02110-2804

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patents issued thereon.

Combined Declaration and Power of Attorney
Page 2 of 8 Pages

Full Name of Inventor: GILBERT WOLRICH

Inventor's Signature: _____ Date: _____
Residence Address: Framingham, MA M A
Citizenship: United States
Post Office Address: 4 Cider Mill Road
Framingham, MA 01701

Full Name of Inventor: MATTHEW J. ADILETTA

Inventor's Signature: _____ Date: _____
Residence Address: Worcester, MA M A
Citizenship: United States
Post Office Address: 20 Monticello Drive
Worcester, MA 01603

Full Name of Inventor: WILLIAM R. WHEELER

Inventor's Signature: _____ Date: _____
Residence Address: Southborough, MA M A
Citizenship: United States
Post Office Address: 9 Darlene Drive
Southborough, MA 01772

Full Name of Inventor: DEBRA BERNSTEIN

Inventor's Signature: _____ Date: _____
Residence Address: Sudbury, MA M A
Citizenship: United States
Post Office Address: 321 Old Lancaster Road
Sudbury, MA 01776

Full Name of Inventor: DONALD F. HOOPER

Inventor's Signature: _____ Date: Dec 11, 2002
Residence Address: Shrewsbury, MA M A
Citizenship: United States
Post Office Address: 19 Main Circle
Shrewsbury, MA 01545

20441424.doc

Combined Declaration and Power of Attorney
Page 2 of 8 Pages

Full Name of Inventor: GILBERT WOLRICH

Inventor's Signature: _[signature]_   Date: June 19, 2002
Residence Address: Framingham, MA
Citizenship: United States
Post Office Address: 4 Cider Mill Road
Framingham, MA 01701

Full Name of Inventor: MATTHEW J. ADILETTA

Inventor's Signature: _[signature]_   Date: June 19, 2002
Residence Address: Worcester, MA
Citizenship: United States
Post Office Address: 20 Monticello Drive
Worcester, MA 01603

Full Name of Inventor: WILLIAM R. WHEELER

Inventor's Signature: _[signature]_   Date: 6/19/02
Residence Address: Southborough, MA
Citizenship: United States
Post Office Address: 9 Darlene Drive
Southborough, MA 01772

Full Name of Inventor: DEBRA BERNSTEIN

Inventor's Signature: _[signature]_   Date: 6/20/02
Residence Address: Sudbury, MA
Citizenship: United States
Post Office Address: 321 Old Lancaster Road
Sudbury, MA 01776

Full Name of Inventor: DONALD F. HOOPER

Inventor's Signature: _[signature]_   Date: June 19, 2002
Residence Address: Shrewsbury, MA
Citizenship:
Post Office Address: 19 Main Circle
Shrewsbury, MA 01545

20441424.doc

POWER OF ATTORNEY
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)

PATENT COOPERATION TREATY
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE

The undersigned applicant:

INTEL CORPORATION

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.
225 Franklin Street
Boston, Massachusetts 02110-2804

☒ as agent(s)   ☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with
the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE
Agent's Reference: 10559-305US1
Application Number: 10/069,229

filed with the United States receiving Office and to make or receive payments on applicant's
behalf.

Place:_____

Date:_____

Signature:_____
          INTEL CORPORATION

Printed Name:_____

Title:_____

POWER OF ATTORNEY
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)

PATENT COOPERATION TREATY
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE

The undersigned applicant:

GILBERT WOLRICH

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.
225 Franklin Street
Boston, Massachusetts 02110-2804

☒ as agent(s)   ☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with
the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE
Agent's Reference: 10559-305US1
Application Number: 10/069,229

filed with the United States receiving Office and to make or receive payments on applicant's
behalf.

Place: _Hudson, MA. US._

Date: _June 19, 2002_

Signature: _Gilbert Wolrich_
GILBERT WOLRICH

Printed Name: _GILBERT WOLRICH_

Title: _Inventor_

POWER OF ATTORNEY
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)

PATENT COOPERATION TREATY
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE

The undersigned applicant:

MATTHEW J. ADILETTA

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.
225 Franklin Street
Boston, Massachusetts 02110-2804

☒ as agent(s)   ☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with
the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE
Agent's Reference: 10559-305US1
Application Number: 10/069,229

filed with the United States receiving Office and to make or receive payments on applicant's
behalf.

Place:_____Hudson, MA_____

Date:_____June 18, 2002_____

Signature:_____
                MATTHEW J. ADILETTA

Printed Name: MATTHEW J. ADILETTA_____

Title: Inventor_____

POWER OF ATTORNEY
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)

PATENT COOPERATION TREATY
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE

The undersigned applicant:

WILLIAM R. WHEELER

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.
225 Franklin Street
Boston, Massachusetts 02110-2804

☒ as agent(s)   ☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE
Agent's Reference: 10559-305US1
Application Number: 10/069,229

filed with the United States receiving Office and to make or receive payments on applicant's behalf.

Place: _____Hudson, MA USA_____

Date: _____6/19/02_____

Signature: _____
WILLIAM R. WHEELER

Printed Name: WILLIAM R. WHEELER

Title: Inventor

POWER OF ATTORNEY
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)

PATENT COOPERATION TREATY
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE

The undersigned applicant:

DEBRA BERNSTEIN

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.
225 Franklin Street
Boston, Massachusetts 02110-2804

☒ as agent(s)   ☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE
Agent's Reference: 10559-305US1
Application Number: 10/069,229

filed with the United States receiving Office and to make or receive payments on applicant's behalf.

Place: _____Hudson MA_____

Date: _____6/20/02_____

Signature: _____Debra Bernstein_____
            DEBRA BERNSTEIN

Printed Name: _DEBRA BERNSTEIN_____

Title: _Inventor_____

POWER OF ATTORNEY
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)

PATENT COOPERATION TREATY
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE

The undersigned applicant:

DONALD F. HOOPER

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.
225 Franklin Street
Boston, Massachusetts 02110-2804

☒ as agent(s)   ☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR ARCHITECTURE
Agent's Reference: 10559-305US1
Application Number: 10/069,229

filed with the United States receiving Office and to make or receive payments on applicant's behalf.
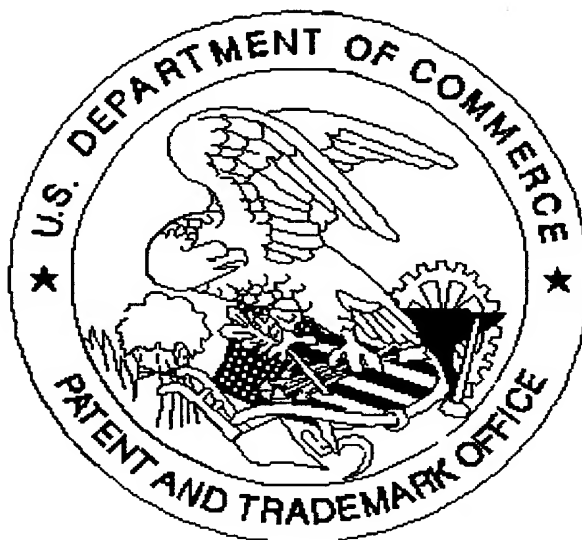
Place:_____*Hudson MA USA*_____

Date:_____*6/19/2002*_____

Signature:_____*Donald F Hoop*_____
DONALD F. HOOPER

Printed Name: DONALD F. HOOPER

Title: Inventor

United States Patent & Trademark Office
Office of Initial Patent Examination -- Scanning Division

Application deficiencies found during scanning:

☑ Page(s) 3/5 to 5/5 of _declaration_ were not present
for scanning.

(Document title)

☐ Page(s)_____ of _____ were not
present
for scanning.

(Document title)

❑ *Scanned copy is best available.*